Birla Institute of Technology and Science Pilani, Hyderabad Campus

2nd Semester 2023-24

25.04.2024

BITS F464: Machine Learning

INSTANCE AND KERNEL BASED LEARNING:k-NN, SVM

Chittaranjan Hota, Sr. Professor
Dept. of Computer Sc. and Information Systems
hota@hyderabad.bits-pilani.ac.in

Instance-based learning: k-NN

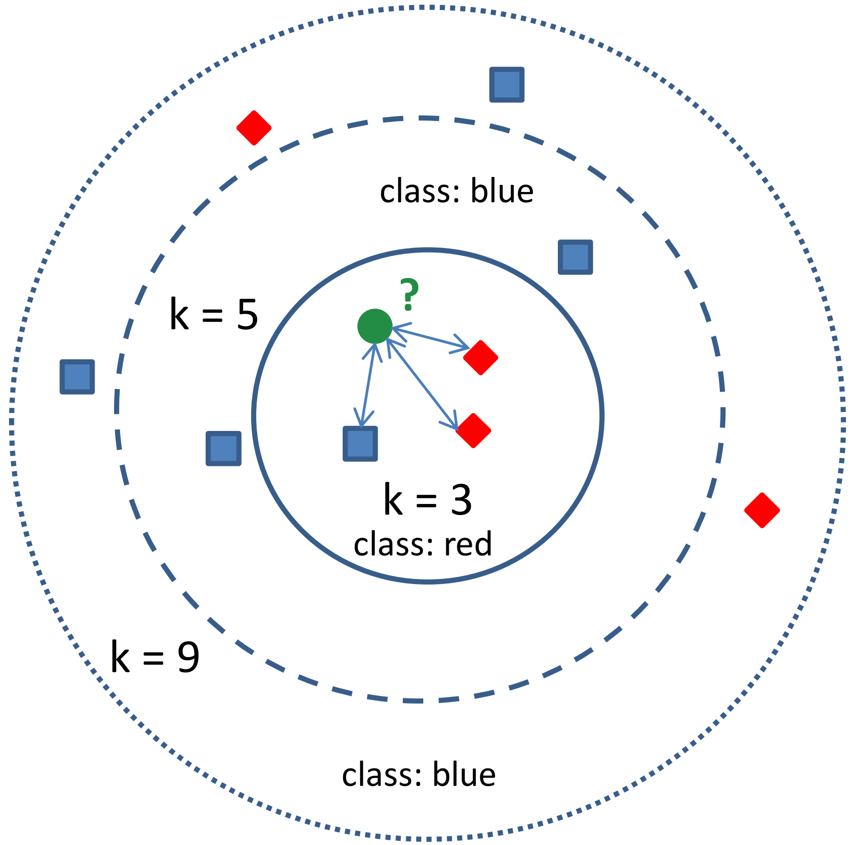
- Why is it called Instance-based?
  - Predictions are made based on specific instances or examples from the training data.
  - Instead of learning explicit relationships between features, it learns from memorization of training data.
- Called Lazy learning. Why?
  - Because it postpones generalization until prediction/ classification time.
- Where is it useful over Symbolic or Connectionist learning you have read?
  - Where the underlying relationships between features and labels are complex OR where the dataset is dynamic and constantly evolving.
- They are robust to Concept drift. Why?
  - As they directly adapt to new examples, they are not affected by data distribution over time or change in the characteristics of the target variable.
- Based on **Similarity metrics** (Euclidian, Manhattan, Cosine, etc…)

k-NN: k-Nearest Neighbor Algorithm

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

load dataset

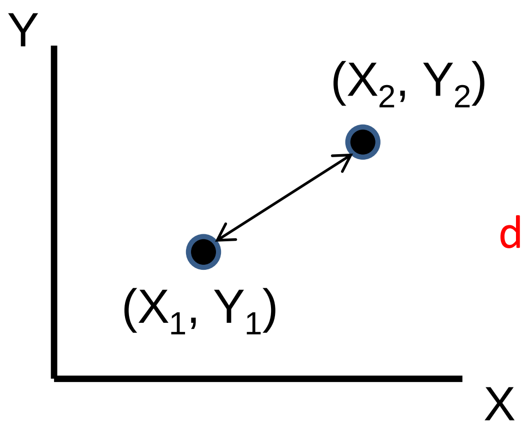
k=5



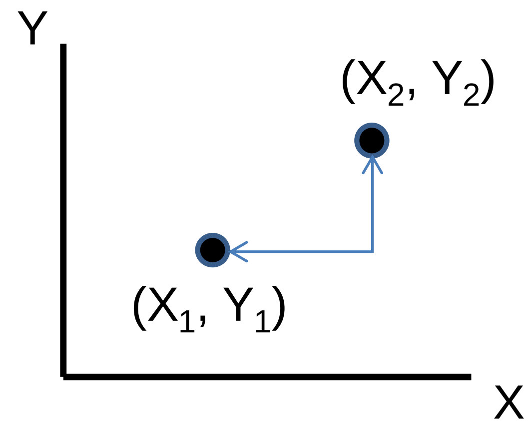
Discrete valued target fun: voting, and
Real-valued target fun: taking mean

Applications: Optical Character Recognition (OCR), Credit Scoring, Loan Approval.

k-NN Distance Metrics: Common Choices



distance: hyper-parameter



Euclidean: $\sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$

Manhattan: $| X_2 - X_1 | + | Y_2 - Y_1 |$

```
from sklearn.metrics.pairwise import
euclidean_distances

point1 = [[1, 2]]
point2 = [[4, 6]]

distance = euclidean_distances(point1, point2)

print("Euclidean Distance:", distance[0][0])
```

```
import numpy as np
point1 = np.array([1, 2])
point2 = np.array([4, 6])

distance = np.sum(np.abs
                  (point1 - point2))

print("Manhattan Dist:", distance)
```

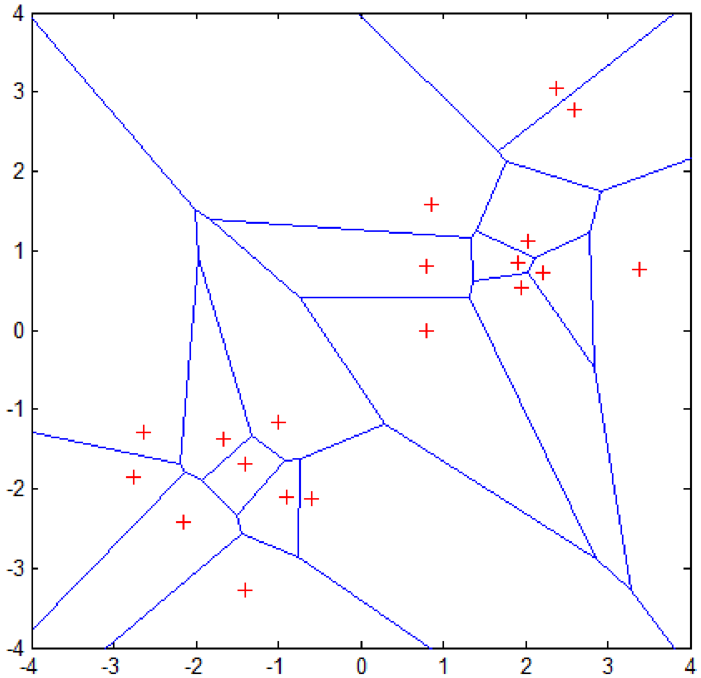
manually

Decision boundaries: Voronoi-like dia.

$$\hat{f}(x_q) = \underset{v \in V}{\arg\max} \sum_{i=1}^{k} \delta(v, f(x_i))$$ Where, δ(a, b) = 1 if a==b, zero (0) otherwise.

Properties:

- All possible points within a sample's Voronoi cell are the nearest neighboring points for that sample.
- For any sample, the nearest sample is determined by the closest Voronoi cell edge
- When you perform a KNN search for a given point, you're effectively partitioning the space around each data point into regions based on distance.



(1-Nearest Neighbor Algorithm)

Distance-weighted k-NN: Refinement

- Weight the contribution of each of the k-neighbors according to their distance to the query point, x_q, giving greater weight to closer neighbors.

$$\hat{f}(x_q) = \underset{v \in V}{\arg\max} \sum_{i=1}^{k} \frac{1}{d(x_i, x_q)^2} \delta(v, f(x_i))$$

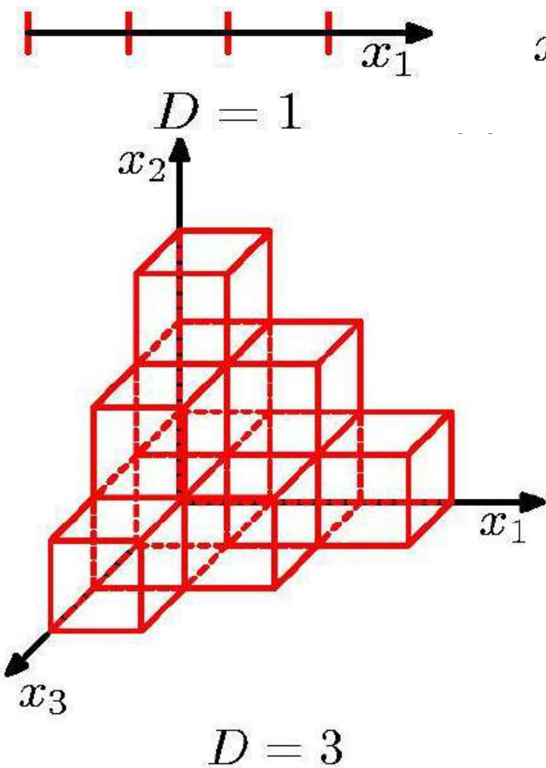
Where, 'd' is the distance between x_i and x_q.

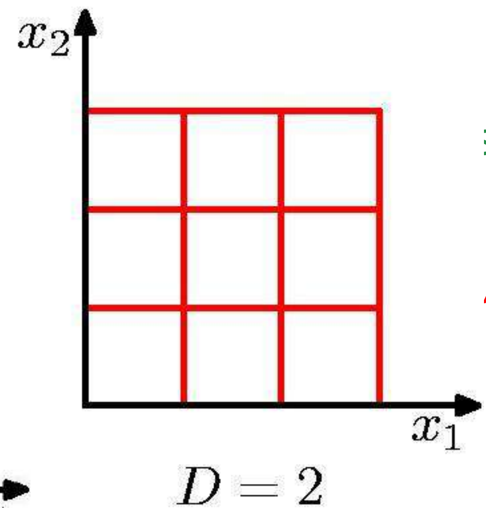
- To accommodate the case where the query point x_q, exactly matches one of the training instances 'x_i' and the denominator $d(x_i, x_q)^2$ will therefore be zero, and hence we assign:

$$\hat{f}(x_q) = f(x_i)$$

- If there are several such training examples, we assign the majority classification among them.
- Closer neighbors have a greater influence on the decision, while farther neighbors have less influence. Sensitive to outliers.

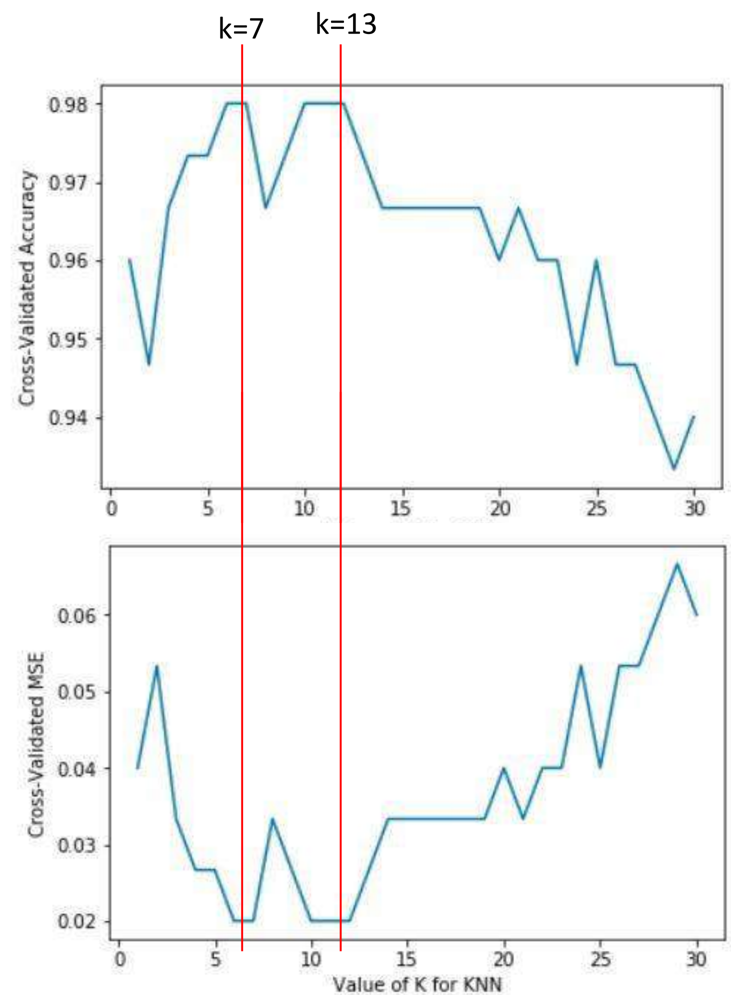
How to choose a right value of 'k'?





er decision boundary
dividual neighbor is

- Reduce the impact of noise and outliers.
- Can lead to Under-fitting in complex cases.
- Hence, High bias and Low variance.

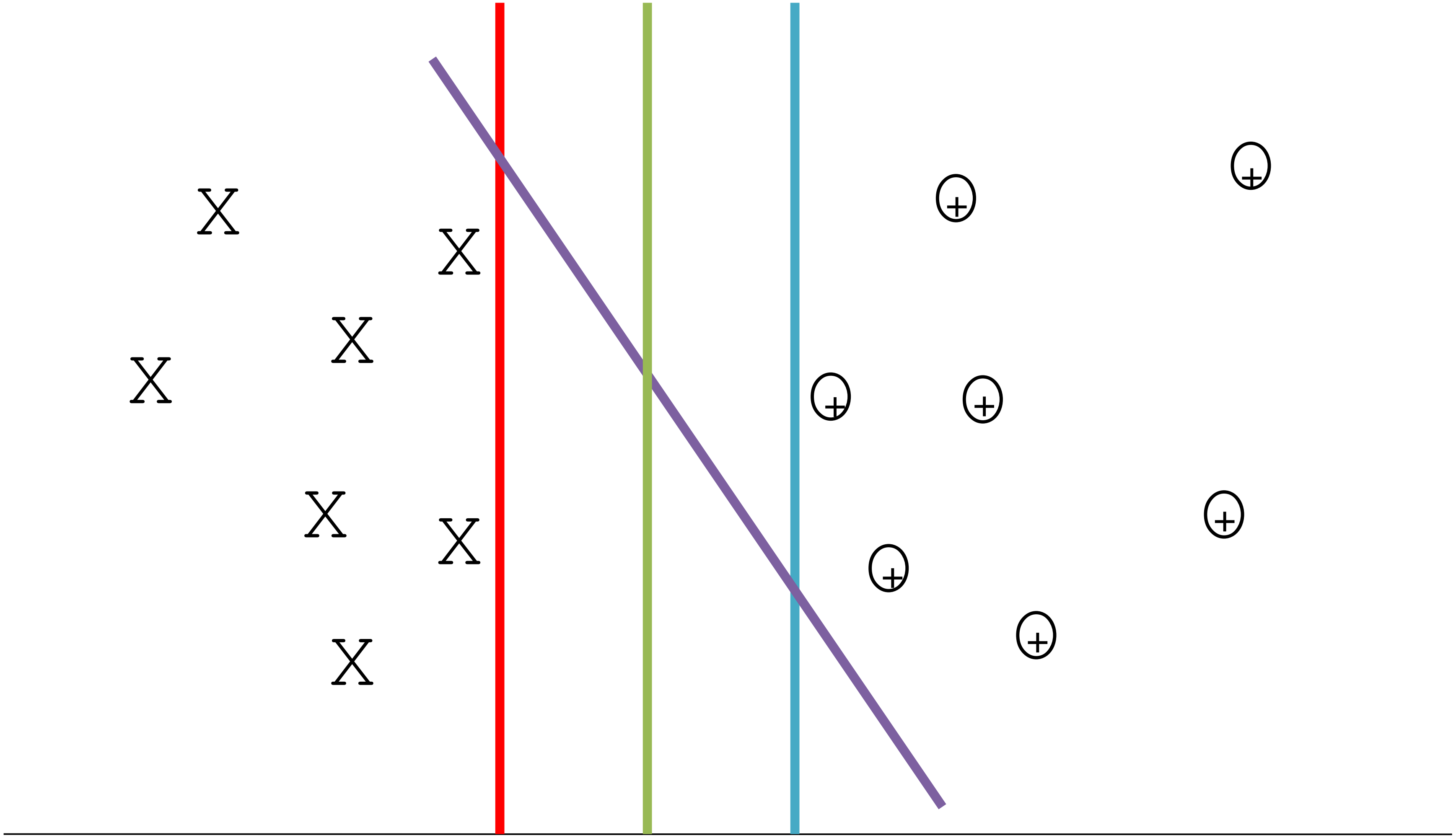


Curse of Dimensionality: Distance computation complexity, Sparse data, Curse of Proximity

Kernel-based Learning: Support Vector Machine

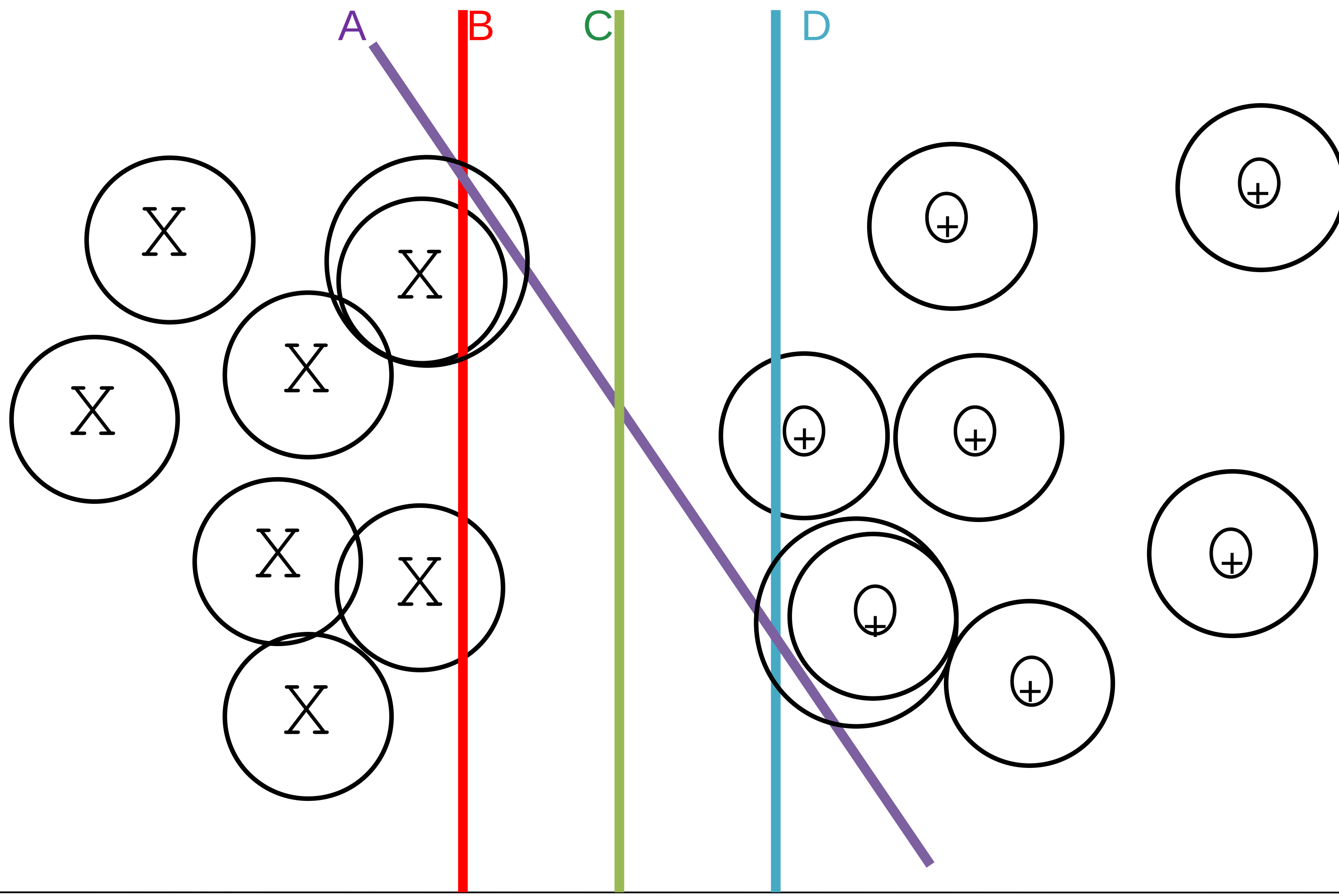
- Transform the input data into a higher-dimensional feature space using a kernel function.
- In this higher-dimensional space, the data may become more linearly separable, allowing linear algorithms to do the classification well.
- SVM finds an optimal hyperplane that separates the classes in this transformed feature space.
- The optimal hyperplane is the one that Maximizes the Margin between the two classes of data points.
- Model complexity depends on the number of training samples, not on the dimensionality of the kernel space.
- As it can handle high dimensional vector spaces with ease, it makes feature selection less critical.

SVM: Intuition behind choice of surface



Which one is the best separator out of these 4?

Some Noise in the Input Samples

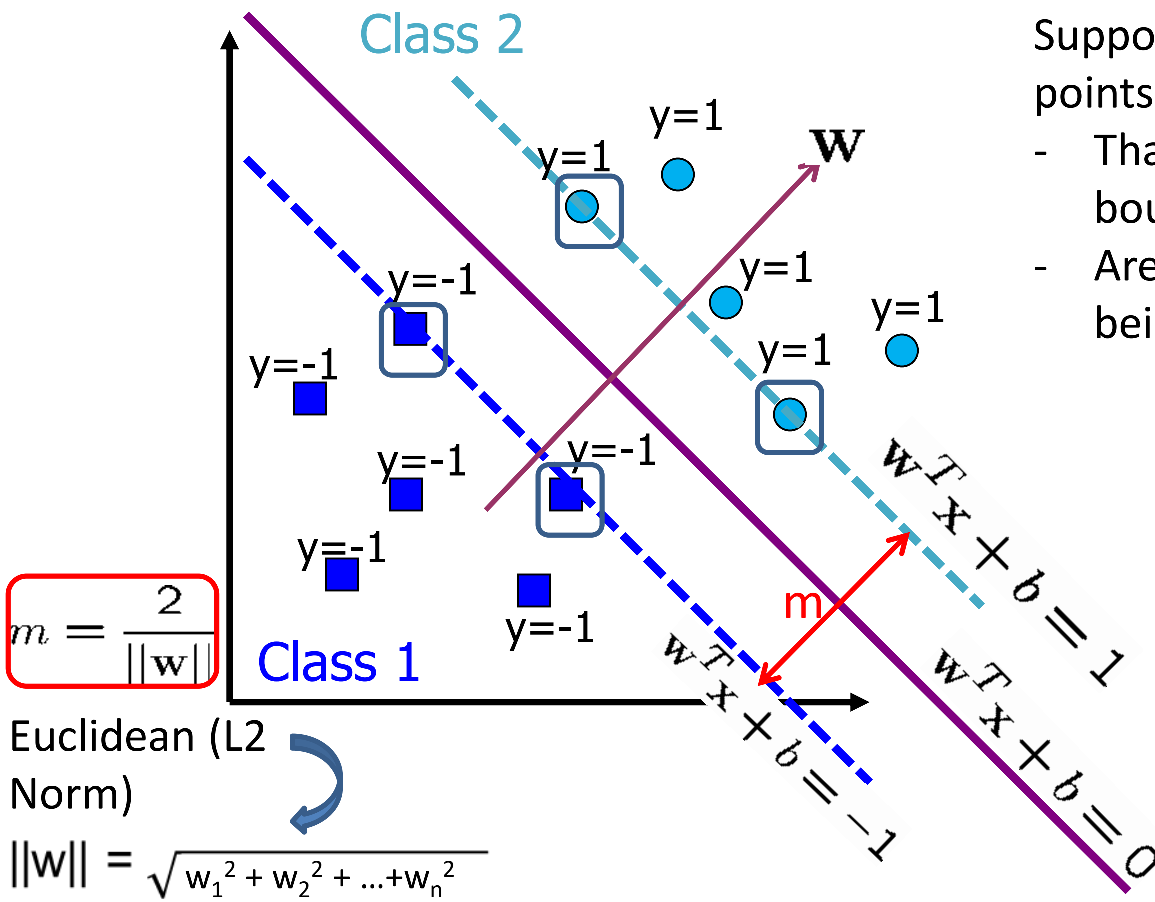


Which ones are better now?

Which one is good ow?

Finding the Decision Boundary: Another Ex.

- Let $\{x_1, ..., x_n\}$ be the data set and let $y_i \in \{1,-1\}$ be the class label of x_i



Support Vectors are the data points:

- That lie on or within the margin boundary.
- Are misclassified or are close to being misclassified.

Hence, the decision boundary should be as far away as possible from the data of both the classes → Maximum Margin Classifier (m)

Support vectors are the data points that lie closest to the decision boundary (hyperplane) and have the largest influence on determining the position and orientation of the boundary.

Maximum Margin Classifier (m)

For the marginal plane, we can write the equation as:
$$w^T x + b = 0$$

For the positive hyperplane the equation will be:
$$w^T x + b \geq 0 \text{ when } y_n = +1$$

And for negative hyperplane:
$$w^T x + b < 0 \text{ when } y_n = -1$$

Marginal Distance?
$$w^T x_1 + b - w^T x_2 + b = 1 - -1$$

$$w^T (x_1 - x_2) = 2 \quad \text{———} \quad \text{(Eq.1)}$$

As w^T is a vector which has a direction, divide the equation (1) by ||w||:
$$\frac{w^T}{\|w\|}(x_1 - x_2) = \frac{2}{\|w\|}$$
$$\text{ie,}(x_1 - x_2) = \frac{2}{\|w\|}$$

Hence, the goal of SVM is:

$$\boxed{\max \frac{2}{\|w\|}} \longrightarrow \text{Regularizer}$$

$$\text{subject to} \qquad y_n(w^T x + b) \geq 1$$

$$\boxed{y_n \begin{cases} +1 & w^T x + b \geq 1 \\ -1 & w^T x + b \leq -1 \end{cases}}$$

Soft margin SVM

- For performing optimization using gradient descent the regularizer can also be rewritten as follows:

$$max\frac{2}{||w||} = max\frac{1}{||w||}$$
$$= min||w|| = min\frac{1}{2}||w||^2$$

$$\underbrace{\min \frac{1}{2}\|w\|^2}_{\text{regularizer}} + \underbrace{C\sum_{i=1}^{n}\xi_i}_{\text{error term}}$$

- Including the number of errors in the training(C) and the sum of the value of error (Σζ), the optimization term will be:

- This term allows some classification errors to occur for avoiding overfitting of our model, i.e, the hyperplane will not be changed if there are small errors in classification.

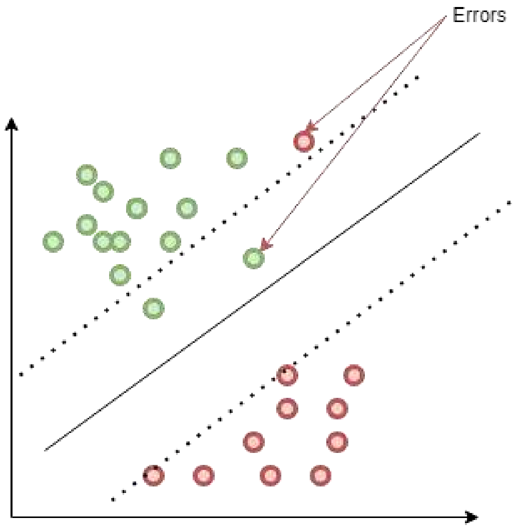


Image source : https://www.pycodemates.com/

Constrained Optimization Problem: Dual

- Provides computational advantages, especially when dealing with large datasets.
- We maximize the expression with respect to the Lagrange multipliers α_i, subject to the constraints.
- This formulation allows for the solution to be expressed entirely in terms of the inner products of the input vectors x_i, which is computationally advantageous, especially when using kernel tricks to map the data into higher-dimensional feature spaces.

Minimize $\|\mathbf{w}\| = \langle \mathbf{w} \cdot \mathbf{w} \rangle$ subject to $y_i(\langle \mathbf{x}_i \cdot \mathbf{w} \rangle + b) \geq 1$ for all i

Lagrangian method : maximize $\inf_{\mathbf{w}} L(\mathbf{w}, b, \alpha)$, where

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\| - \sum_i \alpha_i \left[\left(y_i(\mathbf{x}_i \cdot \mathbf{w}) + b \right) - 1 \right]$$

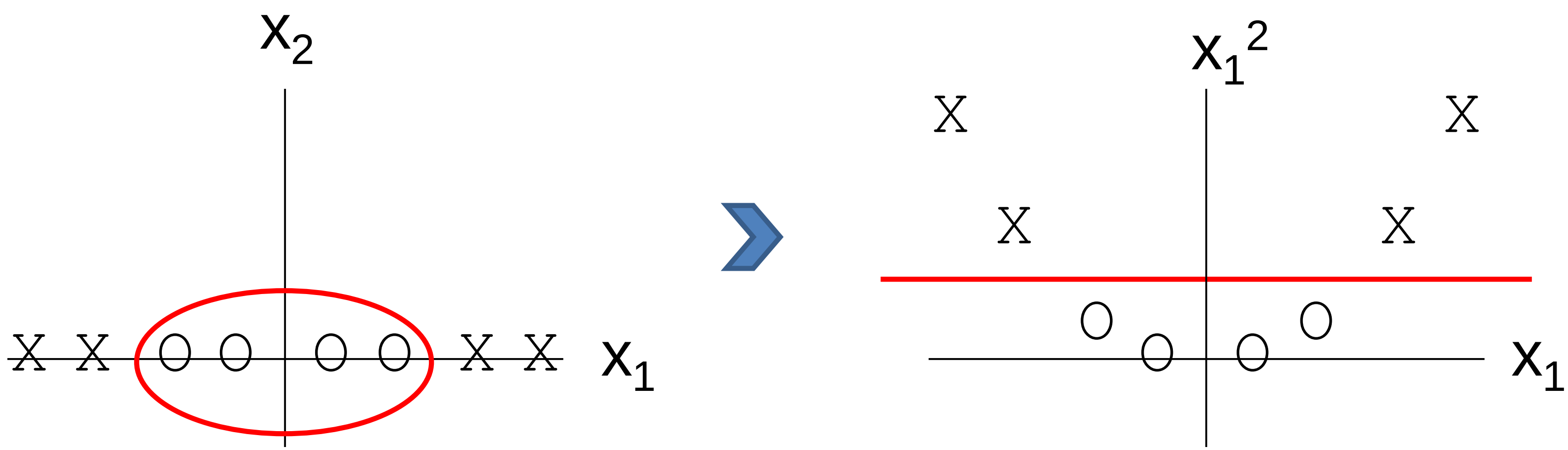
At the extremum, the partial derivative of L with respect both $\mathbf{w}$ and b must be 0. Taking the derivative s, setting them to 0, substituti ng back into L, and simplifyin g yields :

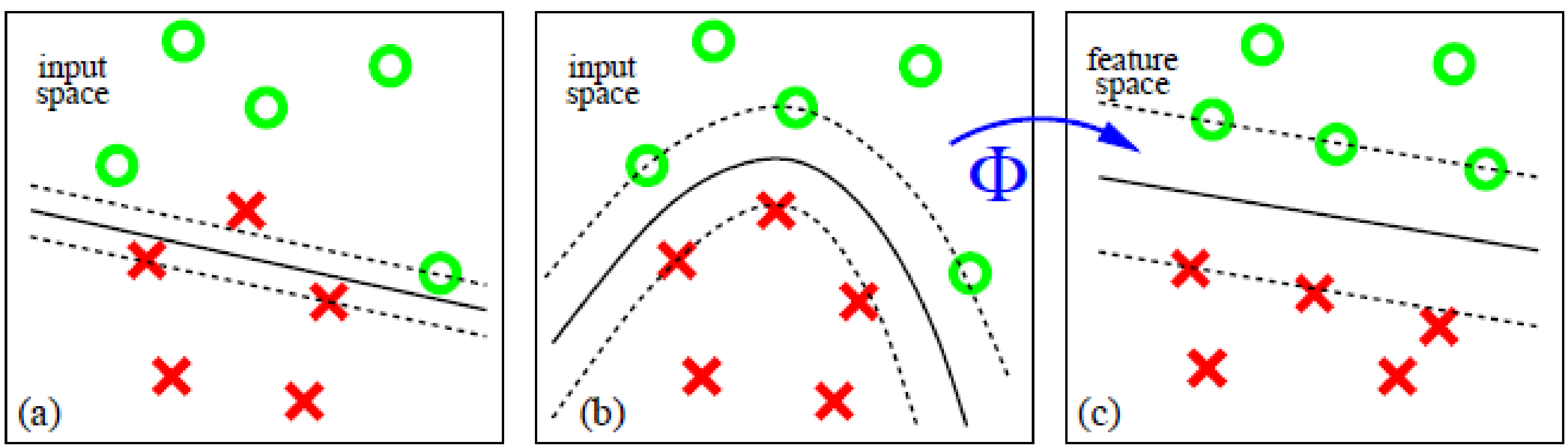
$$\underset{W(\alpha)}{\text{Maximize}} \ \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle$$

subject to $\sum_i y_i \alpha_i = 0$ and $\alpha_i \geq 0$

$$\boxed{\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i}$$

Problems with linear SVM: How to Solve?

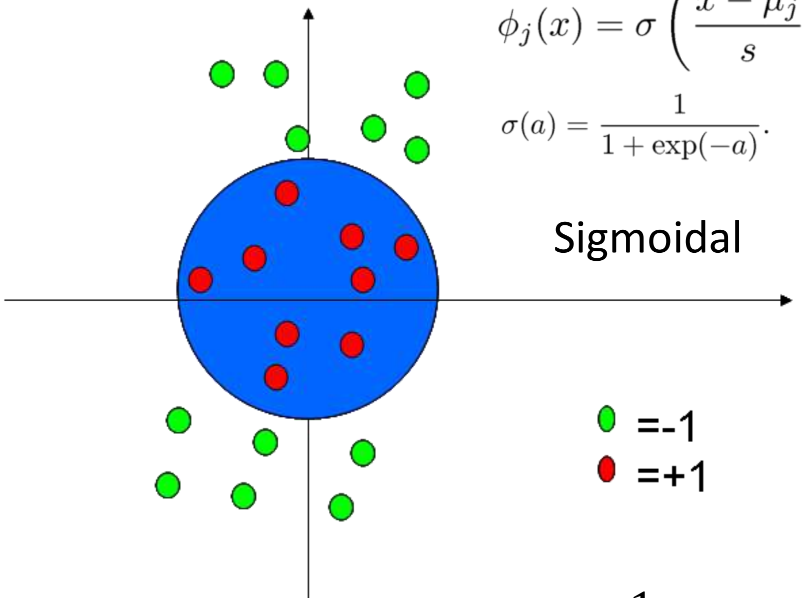




What if the decision function is not a linear?

Given an algorithm which is formulated in terms of a positive definite kernel K_1, one can construct an alternative algorithm by replacing K_1 with another positive definite kernel K_2.

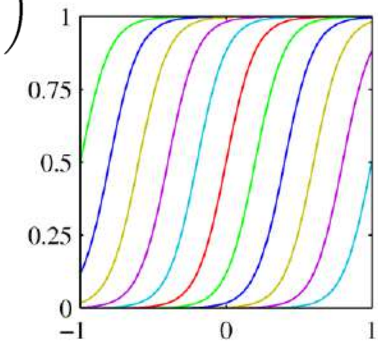
Kernel Trick

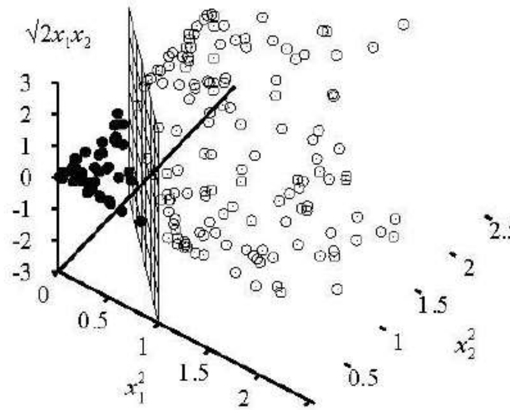


$$\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)$$

$$\sigma(a) = \frac{1}{1+\exp(-a)}.$$

Sigmoidal





Data points are linearly separable in the space $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$

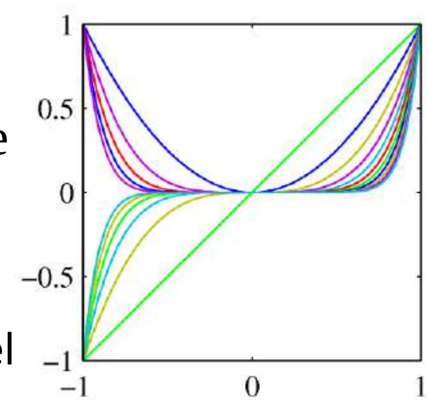
We want to maximize $\sum_i \alpha_i - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j \langle F(\mathbf{x}_i)\cdot F(\mathbf{x}_j)\rangle$

Define $K(\mathbf{x}_i, \mathbf{x}_j) = \langle F(\mathbf{x}_i)\cdot F(\mathbf{x}_j)\rangle$

Cool thing : K is often easy to compute

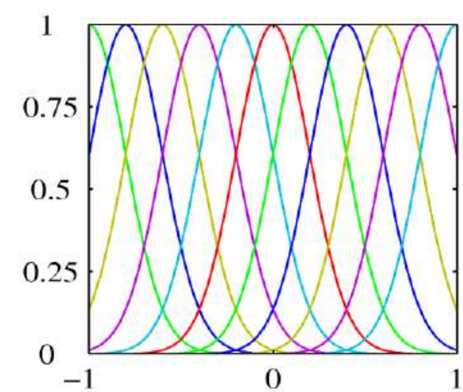
$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle^2$

$K(x_i,x_j) = (x_i \bullet x_j + 1)^p$ polynomial kernel



Gaussian kernels

$K(x_i,x_j) = \exp(||x_i-x_j||^2/2\sigma^2)$



Radial Basis

Radial Basis Kernel: Similar to Gaussian

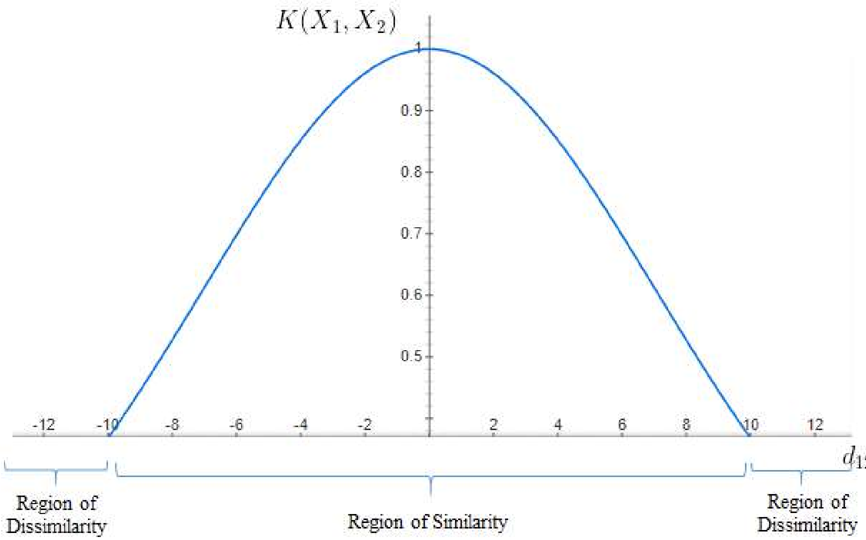
- The RBF kernel function for two points X_1 and X_2 computes the similarity or how close they are to each other. This kernel can be mathematically represented as follows:

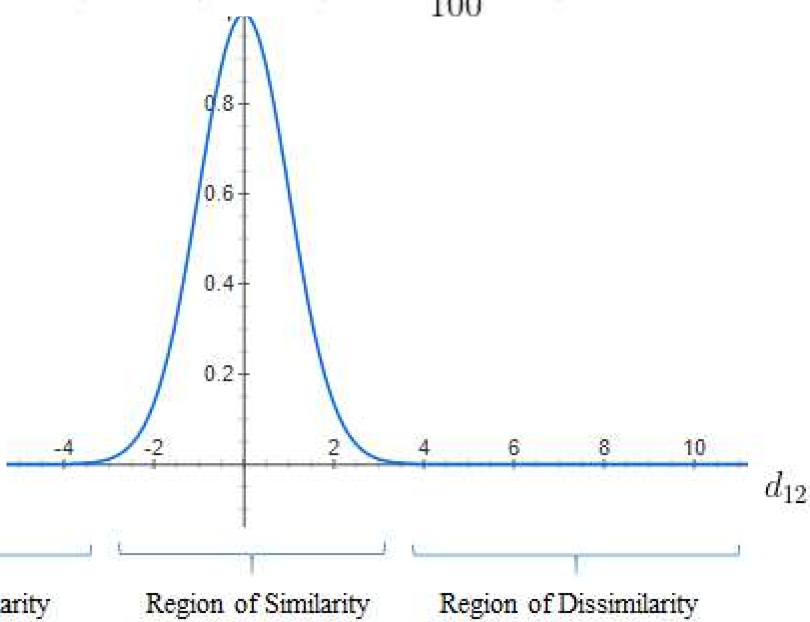
$$K(X_1, X_2) = exp(-\frac{||X_1 - X_2||^2}{2\sigma^2})$$

1. 'σ' is the variance and our hyper-parameter
2. $||X_1 - X_2||$ is the Euclidean (L_2-norm) Distance between two points X_1 and X_2

σ = 10

$$K(X_1, X_2) = exp(-\frac{||X_1 - X_2||^2}{100})$$





$$K(X_1, X_2) = exp(-\frac{||X_1 - X_2||^2}{2})$$

σ = 1

Image source: https://towardsdatascience.com/

Thank You!